Arduino **S**ervice **I**nterface **P**rotocol V1.2 Reference

Michael Margolis

Last updated 20 July 2023

ASIP Overview

Arduino Service Interface Protocol (ASIP) is a protocol that provides communication between a computer and a microcontroller, typically Arduino.

ASIP enables a computer to discover, configure, read and write the microcontroller's general purpose IO pins. In this sense it is similar in spirit to Firmata, but differs in that it is designed to easily expose higher level service that can be implemented on Arduino. The standard ASIP implementation uses a serial connection to the remote computer but other links using a stream interface, such as Ethernet can be used.

Clients send messages to an ASIP server to perform tasks, such as turning on or off pins, controlling servos, generating tones, writing to an LCD screen etc. Clients can also request information from services, such as digital or analog pin values, motor encoder counts, or sensor readings.

Messages that are not service specific are known as system messages. These enable a client to request details on the ASIP version and the microcontroller hardware running the ASIP system.

Microcontroller resources (analog and digital pins) are provided through an IO service that is included with the core ASIP system. Other services are optional and can be included as needed.

The following services are supported with the release of ASIP V1.1:

Service	File to include
Pin level input and output	asipIO.h
Servos	services/asipServos.h
Square wave tone generator	services/asipTone.h
Ultrasonic distance sensor	services/asipDistance.h
LCD display	asipLCD.h
Robot control (motor, encoder)	asipRobot.h
Smart RGB LEDS (neopixels)	asipPixels.h
IMU (Accelerometer, Gyro, Altitude)	asipIMU.h
Heading (compass)	asipHeading.h

ASIP command and event message protocol

1. Definition of the format
    1. Messages consist of an ASCII header, ASCII character fields separated by commas, terminated by the newline character.
    2. Request messages to the server begin with a single character to indicate the desired service followed by a comma and a single character tag to identify the nature of the request.
    3. Requests that contain a parameter are separated from the tag with a comma.
    4. Replay messages from the server begin with one of the following characters:
        a. "@" An event message responding to a request or autoevent. These messages have three bytes following the '@' character, a character indicating the service, a comma, and the tag indicating the request that triggered this event
        b. "~" An error message reporting an ill formed request or some other problem affecting the server. These messages contain the service and tag associated with the error followed by an error number and error string.
        c. "!" An informational or debug message consisting of unformatted ASCII text terminated by the newline character.
    5. Some reply event messages have a payload with a variable number of fields with the following format:
        a. A numeric value precedes the message body indicating the number of fields in the body.
        b. Curley brackets are used to indicate the start and end of fields in the message body
        c. If a message contains sub fields, these are separated by a colon (see analog pin mapping for an example)
        d. All numeric values are decimal ASCII text digits unless otherwise stated.

New in ASIP version 1.2:

- Added support for ESP32 AND Pico2040 boards
- Added support for wider range of LCD hardware, including color displays
- Enhanced ASIP pixels to enable setting of background color on LCD displays
- Added message to update SSID and Pw for boards with build-in WiFi
- Changes are backwards compatible – all ASIP 1.1 clients should work as is in V1.2

New in ASIP version 1.1:

- Support for wider range of motor drivers.
- New motor functions to control robot speed and rotation using PID. These functions are suitable to ROS enable a robot.

Backward compatibility

With one exception, existing ASIP clients should run unaltered with the new version, encoder events in this version is handled differently. The Encoder event interval is now hard coded within the motor service module. Encoder events can be turned on and off using ASIP protocol but the interval cannot be changed (without recompiling the motor source code). Encoder events are now part of the motor service, therefor the encoder event now uses the motor service ID 'M' rather than 'E'. The message fields have also changed to contain the number of pulses since the last message and the total pulse count. Previously the fields were pulse width in microseconds and number of pulses

since last message. A new message has been added to the protocol to enable a client to reset the total pulse count to zero

System Messages from client

Get Version info

Request:

Header	Separator	Tag	Terminator
'#'	,	'?'	'\n'

Reply:

Header		Tag		Major Version		Minor Version		Micro controller		Pins		Sketch Name	Term
'@#'	,	'?'	,	Numeric digits	,	Numeric Digits	,	Text	,	Numeric Digits	,	Text	'\n'

Example:

Request: "#,?\n"

Reply: "@#,?,0,2,ATmega328P,20,TestIO\n"

In this example, ASIP version 0.2 is running on an ATmega328P with 20 pins using a sketch named TestIO:

Get Pin Service List

Provides a list of service IDs associated with each pin

Request:

Header	Separator	Tag	Terminator
'#'	,	'S'	'\n'

Reply:

Header		Tag		Pin Count		Start of Body	Body	End of Body	Term
'@#'	,	'S'	,	Numeric digits	,	'{'	Comma separated characters indicating service ID	'}'	'\n'

Pin count indicates the number of pins on the server. The body will contain a character indicting the service corresponding to a pin for each pin from 0 to pin count-1.

Example:

Request: "#,S\n"

Reply: "@#,S,20,{@,@,I,I,I,I,I,I,I,I,I,I,I,I,I,I,I,I,S,D}\n"

In this example, pins 0 and 1 are reserved (by the serial connection to the client), pin 18 is used by the servo service, pin 19 is used by the distance service. All other pins are available for use by the I/O service.

Get Service Names

Request:

Header	Separator	Tag	Terminator
'#'	,	'N'	'\n'

Reply:

Header		Tag		Pin Count		Start of Body	Body	End of Body	Terminator
'@#'	,	'N'	,	Numeric digits	,	'{'	Comma separated pairs of colon separated values indicating service ID character and service name	'}'	'\n'

Example:

Request: "`#,N\n`"

Reply: "`@#,N,3,{I:ASIP core IO,S:Servos,D:Distance}\n`"

Three services, 'I' is the id for the ASIP core IO, 'S' is the service id for Servos , 'D' is the id for the distance sensor service.

Set Pin Mode

Set the mode of the given pin

Request:

Header	Separator	Tag	Separator	Pin	Separator	Value	Terminator
'I'	,	'P'	,	Numeric digits	,	Mode (See table)	'\n'

Mode	Value	Comment
INPUT_MODE	1	digitial input
INPUT_PULLUP_MODE	2	digital input with pull-up resistors enabled
OUTPUT_MODE	3	digital output
ANALOG_MODE	4	analog input
PWM_MODE	5	pwm output (analogWrite)
The following modes are managed by the system and cannot be requested		
UNALLOCATED_PIN_MODE	0	Pin has not been allocated to a service
RESERVED_MODE	6	pin is used by the server (typically for serial communication)
OTHER_SERVICE_MODE	7	pin is in use by a service
INVALID_MODE	8	pin is not valid

Reply: none (see Get Pin Mode message later in this document)

Example: Set pin 13 to OUTPUT

Request: "`I,P,13,3\n `"

Digital Write

Set the value of a digital pin to 0 or 1

Request:

Header	Separator	Tag	Separator	Pin	Separator	Value	Terminator
'I'	,	'd'	,	Numeric digits	,	0 or 1	'\n'

Reply: None

Example: Set pin 13 to 1 (HIGH)

Request: "I,d,13,1\n"

Analog Write

Set the PWM value of an analog output (PWM capable) pin

Request:

Header	Separator	Tag	Separator	Pin	Separator	Value	Terminator
'I'	,	'a'	,	Numeric digits	,	0-255	'\n'

Reply: None

Example: Set pin 9 to 128

Request: "I,a,9,128\n"

System Events to client

Error Messages

These messages report an ill formed request or some other problem affecting the server. These messages contain the service and tag associated with the error followed by an error number and error string.

Header	Service id related to this error	Request tag related to error	Error number	Error Text	Terminator
'~'	A valid service id	A valid svc tag	Text digits	Text string describing error	'\n'

Error type	Number
NO_ERROR	0
INVALID_SERVICE	1
UNKNOWN_REQUEST	2
INVALID_PIN	3
MODE_UNAVAILABLE	4
INVALID_MODE	5
WRONG_MODE	6
INVALID_DEVICE_NUMBER	7
DEVICE_NOT_AVAILABLE	8
I2C_NOT_ENABLED	9

Error messages begin with a tilde (~) followed by the service ID, a comma, and the tag associated with the error (typically this is the tag in the request that caused the error). The body of the error message is one of the above numeric error codes followed by a brief text description of the error.

Informational Messages

Header	Message text	Terminator
'~'	ASCII character string	'\n'

An informational or debug message begins with the tilde character and consists of unformatted ASCII text terminated by the newline character.

IO Service Messages from client

Get Port to Pin Mapping

The mapping of analog pin numbers to digital pin numbers can be queried by issuing the following request

Request:

Header	Separator	Tag	Terminator
'I'	,	'M'	'\n'

Reply:

Header		Tag		Pin Count		Start of Body	Body	End of Body	Term
'@I'	,	'M'	,	Numeric digits	,	'{'	Comma separated pairs of colon separated hexadecimal values indicating port number and bit mask.	'}'	'\n'

Example: A board with a 328 chip such as the Uno

Request: "I,M\n"

Reply: @I,M,20,{4:1,4:2,4:4,4:8,4:10,4:20,4:40,4:80,2:1,2:2,2:4,2:8,
2:10,2:20,3:1,3:2,3:4,3:8,3:10,3:20}\n"

This indicates that there are 20 pins. The colon separated pairs in braces indicate the port number and bit mask associated with each pin. In this example, the state of pin 0 is obtained by masking port 4 with 1, pin 1 is obtained by masking port 4 with 2, pin 19 is port 3 masked with 0x20. Note that the port and mask values for this message are hexadecimal (All other ASIP fields use decimal values).

Get Analog Pin Mapping

Provides the mapping of analog pin numbers to digital pin numbers

Request:

Header	Separator	Tag	Terminator
'I'	,	'm'	'\n'

Reply:

Header		Tag		Pin Count		Start of Body	Body	End of Body	Term
'@I'	,	'm'	,	Numeric digits	,	'{'	Comma separated pairs of colon separated values indicating digital pin number followed by the analog pin number	'}'	'\n'

Example: A board with a 328 chip such as the Uno
Request: "I,m\n"
Reply: "@I,m,6,{14:0,15:1,16:2,17:3,18:4,19:5}\n"

This indicates that there are 6 analog pins, digital pin 14 is mapped to analog pin 0, digital pin 15 to analog pin 1 etc.

Get Pin Modes

Provides a list of comma separated values indicating the current mode of each pin.

Request:

Header	Separator	Tag	Terminator
'I'	,	'p'	'\n'

Reply:

Header		Tag		Pin Count		Start of Body	Body	End of Body	Terminator
'@I'	,	'p'	,	Numeric digits	,	'{'	Comma separated value indicating the mode of each pin	'}'	'\n'

Mode	Value	Comment
UNALLOCATED_PIN_MODE	0	Pin has not been allocated to a service
INPUT_MODE	1	digitial input
INPUT_PULLUP_MODE	2	digital input with pull-up resistors enabled
OUTPUT_MODE	3	digital output
ANALOG_MODE	4	analog input
PWM_MODE	5	pwm output (analogWrite
RESERVED_MODE	6	pin is used by the server (typically for serial communication)
OTHER_SERVICE_MODE	7	pin is in use by a service
INVALID_MODE	8	pin is not valid

Example:
Request: "I,p\n"
Reply: "@I,p,20,{6,6,0,0,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0}\n"

In this example, pins 0 & 1 are reserved (they are used for serial communication), and pins 4 & 5 are reserved by a service running on the server, the other pins are all available

Get Pin Capabilities
Enables the capability of each pin to be queried

Request:

Header	Separator	Tag	Terminator
'I'	,	'c'	'\n'

Reply:

Header	Tag		Pin Count		Start of Body	Body	End of Body	Terminator
'@I'	'c'	,	Numeric digits	,	'{'	Comma separated bit mask indicating the capability of each pin	'}'	'\n'

Pin Capability	Mask bit
DIGITAL_IO	1
ANALOG_INPUT	2
PWM_OUTPUT	4

Example: An Uno board (or any board with a 328 chip)
Request: "I,c\n"
Reply : "@I,c,20,{1,1,1,5,1,5,5,1,1,5,5,5,1,1,3,3,3,3,3,3}\n"

This indicates that there are 20 pins, all with digital IO capability, pins 3,5,6,9,10,11 are PWM capable, and pins 14 through 19 have analog input capability. The values are derived from a bitfield defined in asip.h:

Note there are no protocol messages to explicitly request sending of digital or analog values to a client. Digital values for pins in INPUT mode are sent to the client when a change is detected. A separate message is sent for each port that has changed data on a selected pin.

Analog values for pins in ANALOG mode are sent at the interval determined by the **Set Autoevent Period** message that is described in the next section

Set Autoevent Period

Sets the period in milliseconds for events to be reported to the client. The maximum interval is just over one minute (65535 milliseconds). A period of zero will turn off autoevents for the given service. Note services with a fixed autoevent period such as Motor encoders will enable autoevents on any positive value.

Request:

Header		Tag		Period	Terminator
'I'	,	'A'	,	Numeric value in milliseconds	'\n'

Reply:
The format of the autoevent reply messages are service specific. The following is the format for autoevents from the core IO service

Header		Tag		Analog Pin Count	Start of Body	Body	End of Body	Terminator
'@I'	,	'A'	,	Numeric digits	'{'	Comma separated pairs of colon separated values indicating analog pin number followed by the analog reading on that pin	'}'	'\n'

Example: request selected analog pin data every 20 milliseconds
Request: `"I,A,20\n"`
Reply: (TODO)

Example turn off scheduled messages:
Request: `"I,A,0\n"`
Reply: none – no autoevent messages will be sent for the I/O service after this request is received

Digital Port Data

Change of state of all pins set as digital input is reported using this message. No explicit request is required; setting a pin to digital input automatically enables its status to be reported.

Request: none

Reply:

Header		Tag		Port		Port bit mask	Terminator
'@I'	,	'd'	,	Numeric digits	,	Hex bit mask indicating state of all digital input pins on this port	'\n'

Example:

The mapping between the value sent in this message and the pins or pins that have changed state is dependent on the chip the ASIP server is running on. This information is provided by the port to pin mapping message (defined above). For the ATmega328, the mapping for the twenty digital pins will be:

```
Pins 0-7:   4:1,4:2,4:4,4:8,4:10,4:20,4:40,4:80
Pins 8-13:  2:1,2:2,2:4,2:8,2:10,2:20
Pins 14-19: 3:1,3:2,3:4,3:8,3:10,3:20
```

`"@I,d,4,10":` This message indicates that digital pin 4 is HIGH and pins 0-3 and 5-7 are LOW
`"@I,d,4,F":` This message indicates that digital pins 0-3 are LOW and 4-7 are HIGH
`"@I,d,2,11":` This message indicates that digital pin 8 and 11 are HIGH, 9,10,12, and 13 are LOW

Configure SSID and Password for boards with built-in WiFI

Header		SSID		Password	Terminator
$$	|	string	|	string	'\n'

Strings should contain between 2 and 31 characters

The system and core I/O services are always available on every ASIP implementation. ASIP provides a number of optional higher level services that can be included in a sketch. These services are built using pre-existing libraries for the hardware as needed to implement the service. The client refers to these higher level services using abstract IDs derived from an enumeration, not specific pin numbers. This decoupling of specific pins enables microcontroller boards to be changed without modifying client software.

The following is a description of the protocol used for currently supported high level services.

Servo

Write a value in degrees to a servo

Request:

Header		**Tag**		**Servo Id**		**Angle**	**Terminator**
'S '	,	'W'	,	Numeric Digits	,	Numeric Digits	'\n'

The Servo Id is an enumeration of servos in the order the servos are instantiated in the ASIP sketch. The first servo instantiated will have an Id of 0, irrespective of the pin used. The second servo will have an ID of 1 etc. In this way, different boards with different pin assignments can be used by the same client without modifying any code.

The angle is a value in degrees and is handled identically to the standard Arduino servo library code.

Reply: None – the server will send an error message if an invalid servo id is sent

Example Request: "S,W,0,90\n" set servo 0 (the first servo) to 90 degrees

Distance

Read an ultrasonic distance sensor. This service can report distance by explicit request or through automatically repeating events similar to the analog value messages (see Set Autoevent Period described in the IO section of this document).

Request explicit distance measurement:

Header		**Tag**	**Terminator**
'D '	,	'M'	'\n'

Request distance autoevents:

Header		**Tag**		**Period**	**Terminator**
'D '	,	'A'	,	Numeric value in milliseconds	'\n'

Reply:

Header	**Tag**		**Distance in CM**	**Terminator**
'@D '	'M'	,	Numeric digits	'\n'

Tone

Sound a tone of a given frequency in Hz and duration in milliseconds. This service is non-blocking, applications that sound a sequence of tones need to explicitly delay for the duration of each tone before sending the next.

Request:

Header		Tag		Frequency		Duration (milliseconds)	Terminator
'T '	,	'P'	,	Number	,	Number	'\n'

Reply: None

Example Request: "T,P,440,250\n" play a 440 hertz tone for 250 milliseconds

RGB LED Pixels

Service to set one or more RGB LEDs, such as WS2812 (neopixels). LEDs are identified by a sequence position number, the first pixel is position 0. Colors are 32 bit packed RGB values. To following can be used to convert individual RGB values to color value: `((uint32_t)r << 16) | ((uint32_t)g << 8) | b`

Request: Set pixels using colon separated list of pairs of pixel numbers and colors.

Header		Tag		Strip Index		Number of Fields		Start of Body	Body	End of Body	Terminator
'P '	,	'P'	,	Nbr	,	Nbr	,	'{'	Comma separated pairs of colon separated pixel positions and32 bit color values	'}'	'\n'

Example: on the first strip, set the first pixel green and third pixel blue

Request: `"P,P,0,2,{0:128,2:16}\n"`

Request: Set pixels using RGB values

Header		Tag		Strip Index		Number of Pixels		Start of Body	Body	End of Body	Term
'P '	,	'p'	,	Nbr	,	Nbr	,	'{'	Semicolon separated groups of pixel index: followed by RGB values	'}'	'\n'

Example: on the first strip, set the first pixel red, second green, third green

Request: `"P,p,0,3,{0:128,0,0;1:0,128,0;2,0,0,128}n"`

Request: Set pixels in sequence – a more efficient message when setting adjacent pixels

Header		Tag		Strip Index		Number of Pixels		First Pixel		Start of Body	Body	End of Body	Term
'P '	,	'S'	,	Nbr	,	Nbr	,	Nbr	,	'{'	Comma separated pairs of colon separated pixel positions and32 bit color values	'}'	'\n'

Request: Set overall brightness

Header	Separator	Tag		Strip Index		Number of Pixels	Terminator
'P '	,	'B'	,	Number	,	Brightness	'\n'

Request: Get pixel info – this will request a message from the server with the number of pixels for the given strip. This value is the maximum number of pixels that can be controlled on the strip and is determined by the value given in the sketch `asipPixel.begin` method. ASIP cannot determine how many actual pixels are connected.

Header	Separator	Tag		Strip Index	Terminator
'P '	,	'I'	,	Number	'\n'

Reply: Info message response

Header		Tag		Strip Index		PixelSize	Term
'@P '	,	'I'	,	Numeric digits	,	Numeric digits	'\n'

See pixelTest.ino or mirtoWifiOutreach.ino for code examples using new begin() method supporting callback function that enables setting color in a supported device such as a color LCD.

Motor

This service provides control of motor power, speed and direction. The motor Id is an enumeration of motors in the order instantiated in the ASIP sketch. The first motor instantiated will have an Id of 0, irrespective of the pins used or type of motor. Motor power is a percent from 0 to 100. Positive numbers turn the motor in one direction, negative numbers turn in the opposite direction.

Request: set the power for a given motor

Header		Tag		Motor Id		Motor power %	Terminator
'M '	,	'm'	,	Numeric Value	,	Numeric Value	'\n'

Reply: None - the server will send an error message if an invalid id is sent

Examples:

`"M,m,0,50"` Set the first motor power to 50%

`"M,m,1,-50"` Set the second motor power to 50% (running in opposite direction)

`"M,m,0,0"` Stop the first motor

Request: set power for both motors

Header		Tag		First Motor %		Second Motor %	Terminator

'M '	,	'M'	,	Numeric Value	,	Numeric Value	'\n'

Reply: None

Examples:

"M,M,30,-30" Set first motor power to 30%, second to -30%

"M,M,0,0" Stop both motors

Request: set Motor RPM (using PID)

Header		Tag		Motor Id		RPM		Duration (ms)	Terminator
'M '	,	'r'	,	Numeric Value	,	Numeric Value	,	Numeric Value	'\n'

Example:

"M,r,1,30,2000" set motor 1 RPM to 30 for 2 seconds (2000 milliseconds);

Note: motor rpm can be changed or motor stopped prior to duration timeout (function is non-blocking)

Request: set RPM for both motors (using PID)

Header		Tag		First Motor RPM		Second Motor RPM		Duration (ms)	Terminator
'M '	,	'R'	,	Numeric Value	,	Numeric Value	,	Numeric Value	'\n'

Example:

"M,R,30,30,1000" set both motors RPM to 30 for 1 second

Note: motor rpm can be changed or motor stopped prior to duration timeout (function is non-blocking)

Request: set Robot Speed in cm per second

Header		Tag		Speed cm/sec		Duration (ms)	Terminator
'M '	,	'c'	,	Numeric Value	,	Numeric Value	'\n'

Example: "M,c,-20,2000" move robot backward at 20 cm per sec for 2 seconds:

Request: Rotate Robot by the given Angle at the given speed

Header		Tag		Speed (degrees per second)		Angle (degrees)	Terminator
'M '	,	'a'	,	Numeric Value	,	Numeric Value	'\n'

Example: "M,a,90,-180" rotate 180 degrees ccw in 2 seconds:

Request: Enable or disable repeated reading and sending of encoder counts for all encoders

Header		Tag		Enable	Terminator
'M '	,	'A'	,	1 enables, 0 disables	'\n'

Reply:

Header		Tag		Number of Encoders		Start of Body	Body	End of Body	Term
'@M '	,	'e'	,	Numeric digits	,	'{'	Comma separated pairs of colon separated values indicating current and total pulse counts for each encoder	'}'	'\n'

Example: `"M,A,1\n"` request continuous encoder data
Reply: `"@M,e,2,{10:100,11:120}\n"`
In this example, the first encoder had 10 pulses since the last event and 100 pulses since the last reset. The second encoder had 12 pulses and a total of 120 since last reset.
pulse widths are 3000 & 3100 microseconds, step counts are 11 & 12.

Request: Reset cumulative encoder counts to zero

Header		Tag		Terminator
'M '	,	'E'	,	'\n'

Example: `"M,E,\n"` reset encoder counts (note commas before and after E)

Acceleration

This service provides 3 axis acceleration values that can be reported by explicit request or in auto-event messages.

Request explicit heading data:

Header	Separator	Tag	Terminator
'A '	,	'M'	'\n'

Request heading autoevents:

Header		Tag		Period	Terminator
'A '	,	'A'	,	Numeric value in milliseconds	'\n'

Reply:

Header		Tag		Number of Fields		Start of Body	X axis		Y axis		Z axis	End of Body	Term
'@A '	,	'e'	,	Numeric digits	,	'{'	Numeric digits	,	Numeric digits	,	Numeric digits	'}'	'\n'

Gyroscope

This service provides 3 axis gyroscope values that can be reported by explicit request or in auto-event messages.

Request explicit heading data:

Header	Separator	Tag	Terminator
'G '	,	'M'	'\n'

Request heading autoevents:

Header		Tag		Period	Terminator
'G '	,	'A'	,	Numeric value in milliseconds	'\n'

Reply:

Header		Tag		Number of Fields		Start of Body	X axis		Y axis		Z axis	End of Body	Term
'@G '	,	'e'	,	Numeric digits	,	'{'	Numeric digits	,	Numeric digits	,	Numeric digits	'}'	'\n'

Compass Heading

This service provides compass heading. This information can be reported by explicit request or in auto-event messages. It is typically implemented using a three axis magnetometer so the auto-event information from this service can also contain the magnetic flux readings in the x,y and z axis.

Request explicit heading data:

Header	Separator	Tag	Terminator
'H '	,	'M'	'\n'

Request heading autoevents:

Header	Separator	Tag		Period	Terminator
'H '	,	'A'	,	Numeric value in milliseconds	'\n'

Reply:

Header		Tag		Number of Fields		Start body	X flux		Y flux		Z flux		Heading	End body	Term
'@H '	,	'e'	,	Numeric digits	,	'{'	Nbr	,	Nbr	,	Nbr	,	Nbr	'}'	'\n'

Pressure/Altitude

This service provides readings for atmospheric pressure, temperature, altitude and heading. This information can be reported by explicit request or in auto-event messages.

Request explicit heading data:

Header	Separator	Tag	Terminator
'P '	,	'M'	'\n'

Request heading autoevents:

Header	Separator	Tag		Period	Terminator
'P '	,	'A'	,	Numeric value in milliseconds	'\n'

Reply:

Header		Tag		Number of Fields		Start of Body	Pressure		Temp		Altitude	End of Body	Term
'@P '	,	'e'	,	Numeric digits	,	'{'	Numeric digits	,	Numeric digits	,	Numeric digits	'}'	'\n'

Tone

Sound a tone of a given frequency in Hz and duration in milliseconds. This service is non-blocking, applications that sound a sequence of tones need to explicitly delay for the duration of each tone before sending the next.

Request:

Header		Tag		Frequency		Duration (milliseconds)	Terminator
'T '	,	'P'	,	Numeric Digits	,	Numeric Digits	'\n'

Reply: None

Example Request: `"T,P,440,250\n"` play a 440 hertz tone for 250 milliseconds

RGBLed

Service to set one or more RGB LEDs, such as WS2812 (neopixels). LEDs are identified by a sequence position number, the first pixel is position 0.

Colors are the RGB 8 bit values packed into a 32 bit integer as follows:

Color = r <<16 + g <<8 + b

Request: Set pixels using colon separated list of pairs of pixel numbers and 32 bit packed RGB color value.

Header		Tag		Number of Fields		Start of Body	Body	End of Body	Term
'P '	,	'P'	,	Numeric digits	,	'{'	Comma separated pairs of colon separated pixel positions and numeric color values	'}'	'\n'

Example Request: `"P,P,1,{0:65280}\n"` set the first pixel to green (255<<8)

Request: Set pixels using colon separated list of pixel numbers and three 8 bit RGB color values.

Header		Tag	Number of Fields		Start of Body	Body	End of Body	Term
'P '	,	'p'	Numeric digits	,	'{'	Comma separated pairs of colon separated pixel positions and numeric RGB color values	'}'	'\n'

Example Request: `"P,p,1,{0:255:0:0}\n"` set the first pixel to red

Request: Set pixels in sequence – a more efficient message when setting adjacent pixels

Header		Tag		Number of Pixels		First Pixel		Start of Body	Body	End of Body	Term
'P '	,	'S'	,	Numeric digits indicating number of pixels in body	,	Numeric digits	,	'{'	Comma separated 32 bit color values for pixels starting from positon given in the First Pixel field	'}'	'\n'

Request: Set brightness of all pixels

Header	Separator	Tag		Brightness	Term
'P '	,	'B'	,	Numeric digits indicating brightness level (0-255)	'\n'

Request: Get pixel info – this will request a message from the server with the number of available pixels.

Header	Separator	Tag	Terminator
'P '	,	'I'	'\n'

Reply: Info message response

Header		Tag		Number of Pixels	Terminator
'@P '	,	'I'	,	Numeric digits	'\n'

LCD

Service to print text and simple graphics on an LCD.

The current implementation supports 5 lines of text with 21 characters per line. This code is memory intensive and is currently only supported on the

Request: write a line of text

Header		Tag		Line Number		Text	Terminator
'L '	,	'W'	,	Numeric digits (top line is 0)	,	ASCII text string	'\n'

Example: `"L,W,0,Hello World\n"` write Hello World to top line

The following

Request: write text starting from given line and column (0 is first line and column)

Header		Tag		Line Number		Column		Text	Terminator
'L '	,	'w'	,	digits	,	digits	'	ASCII text string	'\n'

Example: `"L,w,0,6, again\n"` write again in 7th column of first line

Request: clear screen

Header	Separator	Tag	Terminator
'L '	,	'C'	'\n'

Request: draw a horizontal graph for up to 5 values. Values can range from 0 to 100

As with text, the first line is 0

Header		Tag		Number of values		Start of Body	Body	End of Body	Term
'L '	,	'G'	,	Numeric digits indicating number of values	,	'{'	Comma separated pairs of line numbers:values	'}'	'\n'

Example Request: `"L,G,2,{1:50,3:75}\n"` draw 50% bar on second line, 75% on fourth line